PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F | A2

(11) International Publication Number: WO 99/27432

(43) International Publication Date: 3 June 1999 (03.06.99)

(21) International Application Number: PCT/US98/24711

(22) International Filing Date: 20 November 1998 (20.11.98)

(30) Priority Data:
60/066,742 21 November 1997 (21.11.97) US

(71) Applicant: IBRAIN SOFTWARE, INC. [US/US]; Suite 200, 3977 E. Bayshore Road, Palo Alto, CA 94303 (US).

(72) Inventors: SIKKA, Vishal; 1782 Rocky Mountain Avenue, Milpitas, CA 95035 (US). SIKKA, Digvijay; 1782 Rocky Mountain Avenue, Milpitas, CA 95035 (US). SOARES, Thomas; 34216 Kaspar Terrace, Fremont, CA 94555 (US). PATEL, Sukesh; 1503 Cormorant Court, Sunnyvale, CA 94087 (US).

(74) Agents: CHRISTENSEN, Kory, D. et al.; Fenwick & West LLP, Two Palo Alto Square, Palo Alto, CA 94306 (US).

(81) Designated States: JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: SYSTEM AND METHOD FOR INTEGRATING HETEROGENEOUS INFORMATION

(57) Abstract

A computer-implemented method for querying multiple different types of information, each type of information having a different evaluator, includes receiving a query comprising an identification of at least two evaluators, at least one relationship between the evaluators, and a method of combining results from the evaluators; parsing the query to create an evaluation sequence comprising an ordered sequence of invocations of the evaluators; invoking the evaluators in the evaluation sequence; and combining results from the evaluators according to the method of combining results from the evaluators specified in the query. A system for querying multiple different types of information, each type of information having a different evaluator, includes a parser for receiving a query comprising an identification of at least two evaluators, at least one relationship between the evaluators, and a method of combining results from the evaluators; means, coupled to the parser, for creating an evaluation sequence comprising an ordered sequence of invocations of the evaluators; means, coupled to the creating means, for invoking the evaluators in the evaluation sequence; and means, coupled to the invoking means, for combining results from the evaluators according to the method of combining results from the evaluators specified in the query.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

SYSTEM AND METHOD FOR INTEGRATING HETEROGENEOUS INFORMATION

Field of the Invention

The present invention relates generally to information processing, and more particularly, to a system and method for integrating heterogeneous information.

Identification of Copyright

Description of the Background Art

The key technological challenge in integrating heterogeneous information is that different types of information is analyzed using fundamentally different evaluators. Evaluators perform functions such as searching, prediction, collaborative filtering, data mining, and the like, and include such tools as search engines, neural networks, fuzzy logic-based decision makers, and a variety of other systems well known to those of skill in the art of information processing.

For unstructured text, e.g. news and the like, evaluators based on information retrieval or search based mechanisms work best. For structured database data, evaluators based on data access mechanisms work best. Evaluation mechanisms such as searching, prediction, collaborative filtering, data mining, each have semantic differences.

Structured information is semantically organized and stored in databases, such as relational databases and object oriented databases. Structured information is accessed by semantic retrieval mechanisms with explicit syntax and semantics like SQL, OSQL, and ODBC. The entire database industry including decision support, data access, warehousing, mining, and middleware is built on the ability to store and extract information stored in structured data sources.

Unstructured information, on the other hand, is mostly composed of free form, natural language text, e.g. news articles, documents, messages and web pages. The mechanisms to analyze such information are keyword or concept based search and retrieval. Search engines return either too much irrelevant information or too little relevant information.

Qualitative information is more intuitive in nature. Human responses and experiences such as intuition, market conditions, investment style, personality, preferences or ratings are examples of qualitative information. Techniques for evaluating such information are based on collaborative filtering, qualitative data analysis, at the like.

Quantitative information is evaluated based on precise analytical and mathematical models and expert knowledge. Examples include econometric models in the finance industry for measuring risk and performance as well as predictive models. Examples of the latter include systems for predicting frauds and purchase patterns. In many cases this type of information is proprietary and is used within research departments of organizations.

Unfortunately, conventional systems typically allow querying over only a single type of data. For example, SQL, a standard database query language, provides database access, but not access to unstructured information, qualitative information, or the like. Thus, multiple querying systems are required, increasing complexity and cost.

Likewise, current systems perform only a single type of analysis in a single query. For example, typical analysis mechanisms such as text searches, data access, collaborative filtering, prediction, and the like are performed separately. Thus,

multiple queries are required to achieve multiple forms of analysis, decreasing performance and increasing complexity and cost.

Another disadvantage is that current systems are static. In other words, such systems generally cannot be dynamically extended to incorporate new types of analysis via new evaluators without a substantial and expensive overhaul.

What is needed, then, is a system and method for integrating heterogeneous information in which querying may be performed over multiple types of data in a single query. What is also needed is system and method for integrating heterogeneous information that is dynamically extendible to incorporate new types of anaylsis.

SUMMARY OF THE INVENTION

The present invention overcomes the aforementioned problems by providing a Universal Analysis Language (UAL) and the Universal Analysis Model, which is used by the method and system described herein to query over multiple different types of information using multiple different mechanisms for analyzing and evaluating information.

The information to be queried can be located in multiple sources, for example databases, news wires, text files, online (Web) pages, can be of multiple types, e.g. unstructured text, structured database data, quantitative (numeric) data or qualitative data. The mechanisms to analyze and evaluate such information (hereafter referred to as "evaluators") can be equally diverse, e.g. text search methods to evaluate text data, data analysis methods for structured data, collaborative filtering for qualitative data, prediction for quantitative data, etc. The present invention allows users to query multiple different types of information, using multiple different evaluation mechanisms in a single framework for decision making.

In accordance with one aspect the present invention, a computer-implemented method for querying multiple different types of information, each type of information having a different evaluator, includes receiving a query comprising an identification of

at least two evaluators, at least one relationship between the evaluators, and a method of combining results from the evaluators; parsing the query to create an evaluation sequence comprising an ordered sequence of invocations of the evaluators; invoking the evaluators in the evaluation sequence; and combining results from the evaluators according to the method of combining results from the evaluators specified in the query.

In another aspect of the invention, a system for querying multiple different types of information, each type of information having a different evaluator, includes a parser for receiving a query comprising an identification of at least two evaluators, at least one relationship between the evaluators, and a method of combining results from the evaluators; means, coupled to the parser, for creating an evaluation sequence comprising an ordered sequence of invocations of the evaluators; means, coupled to the creating means, for invoking the evaluators in the evaluation sequence; and means, coupled to the invoking means, for combining results from the evaluators according to the method of combining results from the evaluators specified in the query.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a flow chart of a method for integrating heterogeneous information using a universal analysis language (UAL) in accordance with an embodiment of the present invention.

Figure 2 is an illustration of a plurality of truth functions in accordance with an embodiment of the present invention.

Figure 3 is a screen view of a query in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Universal Analysis Model

The Universal Analysis Model (UAM) is a means to integrate the above-described diverse evaluators within a common framework. The UAM achieves this integration by combining the results of analyzing various bodies of information within

a single framework. In addition, the UAM must ensure that this integration is semantic, in that the results of various evaluators are combined correctly. By correctness, it is meant the logical correctness (or validity) of a query over disparate, heterogeneous, information using different evaluators

A key aspect of the present invention is the Universal Analysis Language, including a model, i.e. a formal interpretation structure, for that language. The Universal Analysis Language universalizes the analysis of information from diverse evaluators.

The declarative syntax of the language has two key characteristics. First, it enables expressing the relationships between various evaluators as well as combining the results obtained from these evaluators. The language employs multi-valued logic, a generalization of fuzzy logic, and an interpreter. It is based on a generalization of dynamically interpreted infinite-valued logic. With this language one can express the relationship between various evaluators and their relationship to the validity of decision criteria given the underlying information. In addition, this language enables one to handle inconsistency and incompleteness in the underlying information, as well as find incomplete or inconsistent information.

Second, the language is dynamically extensible. It includes a mechanism to add as well as execute new evaluation mechanisms (either native evaluators or third-party evaluators) on the fly. Dynamic extendibility is supported the architecture in a variety of ways. In one embodiment, a DLL-based execution mechanism called a "Module Manager" can be dynamically extended (i.e. a new evaluator can be added without recompiling or even restarting the system). This approach is primarily used in, for example, a Microsoft Windows™ environment. In another embodiment, a Platform Independent CORBA based execution mechanism that uses an Evaluator Invocation Repository (based on CORBA's DII - the Dynamic Invocation Interface) is used.

The key idea of the language is to express the relationship between the results of various evaluators, and their relationship to the truth of decision criteria in a user query. In one embodiment, two primary methods for combining criteria include: (i)

Statistical, based on weighted average, and (ii) Intuitive Logic, based on an extension of multi-valued logic (Similar to a generalized version of fuzzy logic) with preferences for individual criteria.

The above-described system provides several distinct benefits and advantages. First, is the ability to query over multiple types of data. For example, the underlying data can be text, structured, qualitative, quantitative. Second, is the fact that multiple types of analyses can be combined in a single query. For instance, the underlying analysis mechanisms can be text search, data access, collaborative filtering, prediction, or other analyses. Third, is dynamically extensibility. New forms of analysis based on new evaluators can be dynamically added to the system without significant cost or disruption. Fourth, is the fact that importance values can be assigned to individual decision criteria. Each criteria in a decision can have an associated importance value. Fifth, is the ability to use partially specified decision criteria. Information in a Criteria can be specified imprecisely. Terms such as "High", "Low", and "Aggressive" may be used instead of precise, domain-specific quantities.

System Implementation

The system and methods disclosed herein may be implemented using a general purposes computer, such as an IBM PC or compatible machine. For example, embodiments of the present system may be implemented using the C++ and Java programming languages and executed on such a computer.

In one embodiment, the implementation infrastructure consists of an object oriented programming language, a lexical analyzer, and a parser. Typical language interpretation systems take one or more files as input. The contents of the input file are logically concatenated and made available to the lexical analyzer. The lexical analyzer analysis the input stream of characters and based on user defined *regular expressions*, outputs a sequence of *tokens*. The sequence of tokens is then made available to the *parser*. The parser uses the grammar defined by the programmer to analyze the input stream of tokens and build a *parse tree* called an *abstract syntax tree*. The abstract syntax

tree is then traversed to perform various operations depending on the needs of the language designer.

Object oriented programming (OOP) technology allows *encapsulation* of both data and algorithms into *Object classes*. OOP allows programmers to associate *methods* (functions or procedures) and *variables* (which may themselves be *instances* of objects) with object classes. Object classes can inherit both behavior (i.e., methods) and data from parent *super-classes*. Classes that inherit properties from parent classes are called sub-classes. In typical OOP languages, a sub-class can be used in any context where its parent class can be used. Object instances are created just as basic primitive types like characters or integers are created in conventional languages. Object instances can be named and used just like any other variables. OOP languages provide special syntax to refer to object instance variable and methods.

In one embodiment of the invention, there is defined a set of Object Classes and Special Functions for defining, using, and combining Criteria in useful ways. For example, one embodiment includes the following components:

1. *Parser*: The parser takes text definitions for one or more Criteria as input. The input is parsed to create a set of Criteria Definition Objects – one for each input text definition.
2. *Object Manager*: The object manager serves as a repository for criteria definition objects, values (such as literals and definitions), and the expressions used to evaluate values for individual Criteria.
3. *Module Manager*: The module manager allows retrieval of external function objects given only the name of the function as input. This component also allows for functions and object classes to be dynamically added to the system.
4. *Virtual Table*: The Virtual Table provides all the available candidate data. This component unifies access to candidate information that may come from various data sources and data types.

5. *Special Functions/Classes*: These classes were developed for the end-user who will define Criteria. These functions and classes help to make the process of developing Criteria much more efficient and easy.

Parser

The parser accepts a stream of Criteria Definitions in text format and builds a set of Criteria Definition Objects. One aspect of the present invention includes a special language called Criteria-Script (UAL) that is tailored to make it extremely easy and efficient to define and use Criteria.

Table 1 provides an overview of the principle classes that are used to build the UAL parser.

Table 1: Implementation Classes for the UAL Parser

Num	Class Name	Description
1	Parser	Created for encapsulating the parser. This class provides a function called "parser" that takes one of more Criteria Definitions and returns a list of Criteria Definition Objects. The BNF (grammar) for UAL (i.e., the language the parser is built to recognize) is described below.
2	CriteriaDefinition	Created to encapsulate a single Criteria Definition. This class has slots (i.e., attributes) to hold data structures for various information items like: (a) the name of the Criteria, (b) its return type (primarily fuzz-truth value or real), (c) parameter list for Criteria, and (d) a CriteriaValue object for each defined *value*.

Table 1: Implementation Classes for the UAL Parser		
Num	**Class Name**	**Description**
3	CriteriaValue	Created to encapsulate the definition for a single value in a Criteria Definition. Stores and provides access to the following information items: (a) name of value, (b) list of parameters to the value, and (c) the expression that defines how to evaluate (i.e., compute) the result for the value.

To facilitate the definition and use of Criteria, a domain specific language called UAL was developed. UAL has the following important features:

1. UAL allows Criteria to be easily defined. Criteria can return fuzzy-truth values or any real (float) valued result.
2. UAL provides conversation operators for mapping between real values and fuzzy-truth values.
3. UAL provides special support for dealing with fuzzy truth-values, fuzzy-expressions (expressions using logical/arithmetic operators that return a fuzzy truth-value.
4. UAL provides special support to access and manipulate information from the Virtual Table mechanisms (see below).
5. UAL provides an efficient mechanism to add functions at run time that may then be immediately used to define new Criteria.
6. UAL is easy to use and has simple intuitive semantics that are tailored to defining and combining Criteria results in a safe and efficient manner.

<u>Object Manager</u>

The object manager (OM) was developed to provide convenient storage and access to data structures (i.e. objects, literal constants, names, etc.) that are necessary to support the implementation mechanisms for defining and evaluating Criteria. The object manager's functionality is available through a application programming

interface (API) (i.e., behavior that can be invoked from external functions, objects, procedures, etc.). Table 2 outlines the key methods supported by the Object manager.

Table 2: Key Methods Supported by the Object Manager		
#	Method	Description
1	OBJMGRInitialize	This method takes a list of CriteriaDefinition objects and stores every element of the list for subsequent access and processing. This method has been designed in a flexible manner. It can either accept the list of CriteriaDefinitions from the "parser" function or some other external mechanism (file, data base, other objects, etc.) that can server as a source for the list of Criteria definitions.
2	OBJMJRGetDefinition	This method accepts a Criteria name as input and returns the corresponding CriteriaDefinition object.
3	OBJMJRGetCriteriaDef	This method returns the definition expression – that is, the method returns the expression object (as a VT_Object – see below). The expression object is the object that encapsulates the expression that the end-user used to define the Criterion as a whole or any particular value definition. The method takes as input either, just the name of the Criterion, or the name of the Criterion and the name of the value. The method returns the corresponding expression object.
4	OBJMGRGetValueDefinition	This method takes as input the name of the Criterion and the name of a value definition in that Criterion. This method returns the corresponding CriteriaValue object.

Module Manager

The Module Manager allows functions to be dynamically added and retrieved to support the execution (i.e., interpretation) of the expressions contained in Criteria. The following classes and functions are used to support the capabilities of the module manager. Table 3 outlines the key sub-components needed to implement the module manager.

Table 3: Key Methods Supported by the Module Manager		
#	Class	Description
1	VT_Functor	This abstract class is used to define a common interface for encapsulating functions. All functions that need to be used in Criteria definitions are "wrapped" in the VT_Functor class. In addition, the code that implements each function must be contained in a dynamically loadable code segment [i.e. Windows DLL, Java class, etc.]; the code segment must define one method named "newFunction" that takes as arguments the name of the function and a set of arguments and returns an instance of the function, which as mentioned above inherits from VT_Functor. The handle can then be used to call a function by name and use it in computing values for expressions. VT_Functor is a sub-class of VT_Base. See the description of VT_Table for more information on VT_Base.
	Function	Description

Table 3: Key Methods Supported by the Module Manager		
#	Class	Description
1	AddObjectImplementation	Takes the name for the object and executable in a dynamic load library format. This function returns no value but it adds the name and the library unit in the internal data structure maintained by the module manager component.
2	RemObjectImplementation	Removes both the object class name and the library executable from the internal data structure of the module manager component.
3	AddFunctionImplementation	This function takes a function name and the name of a code segment that can be dynamically loaded using whatever facilities the OS/Platform provides for this purpose (such as DLLs or Java Classes). This function stores the function name and its executable into the internal data structure maintained by the module manager system.
4	RemoveFunctionImplementation	Removes both the function name and the library executable from the internal data structure of the module manager component
5	NewFunction	Takes as input a valid function name that has been added to the module manager using the AddFunctionImplementation function. This function also accepts as optional list of arguments. This function creates an instance of VT_Functor and returns a handle to the instance. The instance can be used to call and execute the function with its needed input arguments.

Virtual Table

The VT is designed to hold all candidate-related information. Because candidate information can come from multiple sources and be of multiple types the VT was designed as a table whose cells are object classes. The VT table can be accessed given a candidate identifier that must be unique – each row of the VT corresponds to one and only one candidate. The implementation of the VT is supported by the classes listed and described in Table 4.

Table 4: Classes Related to Implementation of the Virtual Table

#	Class Name	Description
1	CandidatePool	This abstract class creates an abstraction that represents the entire set of candidates. This class allows the retrieval of a CandidateSet (see below). The CandidateSet that is needed can be retrieved by specifying members with a specific fields, whether or not null values are needed, etc. The fields/columns of the table desired, and whether it's acceptable to skip rows that have Null values in any or all of the needed fields/columns needs to be specified. This interface/abstract class is implemented by the CandidateTest class.
2	CandidateSet	This class delivers a sub-set of the entire pool of available candidates. CandidateSet provides capabilities to systematically iterate and access each candidate in a sequential manner through iteration.

Table 4: Classes Related to Implementation of the Virtual Table		
#	Class Name	Description
3	VT_Base	The base class (i.e., the root) for all classes related to supporting the Virtual Table mechanism. VT_Base is created to enforce consistent functionality of all sub-classes. In particular, there have been added methods like IsConstant, IsReference, etc., that can be used to find out important attributes of the objects.
4	VT_Float32	This [abstract] class/interface inherits from VT_Base and is created to define a standard interface to floating point numbers. Similarly there have been defined other classes to take care of common data types like integers (VT_INT32) and Strings (VT_String).
5	VT_Fuzzy	A [abstract] class/interface that inherits from VT_Base and defines a consistent interface to fuzz-truth values. Methods are provided to access and set fuzzy truth-values as well as to validate the notion a fuzzy-truth value – a real valued number between zero and one. VT_Fuzzy_Imp a standard implementation of VT_Fuzzy. The VT_Fuzzy interface has been used to define [the argument types accepted by] the fuzzy logical operators: And, Or, Not, and Equality.

Table 4: Classes Related to Implementation of the Virtual Table		
#	**Class Name**	**Description**
6	VT_Set	An abstract interface to a set (i.e. collection) of VT objects. VT_Set inherits from both VT_Base and FilterOperand [see table 6], which allows any object implementing the VT_Set interface to act as the left hand side operand of the filter operator [see table 6]. VT_Set has methods defined to allow you to obtain an iterator that can iterate over the members of the set.
7	VT_Iterator	An abstract interface to an iterator that allows iteration through a set of VT objects
8	VT_Composite	The class VT_Composite is an abstract interface to a VT object that contains a set of named fields, each of which has a VT object value and can be individually retrieved by specifying the field's name. This class inherits from both VT_Base and SelectionOperand, which allows any object that implements VT_Composite to act as left hand side of the selection operator [see table 6].
7	SelectionOperand	See table 6.
8	FilterOperand	See table 6.

The VT table provides a consistent, well-defined, interface for the language execution mechanism to access data and information in flexible ways.

Special Classes/Functions

In one embodiment of the present invention, two types of special Object Classes and Functions are provided:

1. A set of functions to make it easier to write and use Criteria definitions.
2. A set of functions and classes to make the implementation of the system easier and more efficient.

Table 5 lists the functions designed and developed for defining and using Criteria:

Table 5: Functions that make Defining Criteria Easier		
#	**Function**	**Description**
1	PI	This function allows users to convert a floating-point number into a fuzzy-truth value. The function takes a definition for a fuzzy curve whose y-axis corresponds to the truth-value between 0 and 1. For a particular point on the x-axis, the y-axis gives the corresponding truth-value. PI takes an object, generally a variable such as candidate.length, as input and the fuzzy-curve. This curve could be a trapezoid, defined by 4 points, or any arbitrary curve defined by n points, or via an equation that can be evaluated using the module manager.

Table 5: Functions that make Defining Criteria Easier		
#	Function	Description
2	And/Or/Not/Wimp	And, Or, and Not are standard logical operations that can be applied to fuzzy truth-values as well as to Boolean (two-valued) truth-values. Wimp, an acronym for With Importance, is a special operator that can be used by end-users to weight specific Criteria more or less heavily than other Criteria. Wimp is a binary operator. Thus, given an expression like (wimp(a1,w1),wimp(a2,w2)) the result would be (a1*w1 + a2*w2)/w1+w2). One interpretation of A WIMP B is logically equivalent to A OR NOT B.
3	>,<,>=, <=,=,!=	Standard comparison operators. These operators produce a strict/boolean truth value, although the results is of type VT_Fuzzy.
4	StringSimilarity	This function takes two input text strings and returns the extent to which the contents of the two strings are similar. The results for these functions are in the range [0,1].
5	List selection operator to selecting a list of candidates	See the filter operand/Apply Filter stuff in table 6.
6	Average	Average takes an input set and computes the average of the list. By using the module manager it is possible to dynamically load all other important statistical operators such as median, mode, max, min, and standard deviation, among others.

Table 6 lists additional classes and functions that helped to make the implementation easier and more efficient.

Table 6: Classes/Functions to make Implementation Easier and Efficient		
#	Class	Description
1	VT_CriteriaUse	An object class that facilitates the use of existing Criteria definitions to define new Criteria definitions. This class stores the name of the Criteria, the parameters it needs etc. so that it can be evaluated by using the module manager to fetch all needed functions to use the results of Criteria. When an object needs to be evaluated, the name and value name associated with that criteria are passed to the Object manager to obtain the expression that is used to evaluate that particular value.
2	VT_FilterParam	This object encapsulates use of the universal quantifier. For example in the expression ("all ~x in working set where (~x.length > 10) ") ~x is said to be the filter parameter. It ranges over all the candidates in the working set (see Appendix 1 for details of thee working set). ~x.length is the use of a filter parameter – specifically ~x.length is bound to the length of the "current" candidate of the working set. The universally quantified variable ~x is represented by a VT_FilterParam class.

Table 6: Classes/Functions to make Implementation Easier and Efficient		
#	Class	Description
3	VT_RowPlaceholder	UAL expressions can use the special keyword "row". (e.g., working set.[row.Turnover > 100]) refers to the candidates whose Turnover is greater than 100. This function is identical to the function VT_FilterParam, except that its value is always defined by the innermost "for all" that encloses it—i.e. ws[row.length > 10] is identical to (all ~row in ws where (~row.length > 10).
4	VT_WorkingSet	This class holds the current working set. It implements the VT_Set interface. Each member of the set is of type VT_Composite, where the named fields of the VT_Composite object correspond to the named fields of the candidate. The current working set is defined to be some subset of the universe of candidates. The working set in IODF is defined as the set of candidates over which a criteria is evaluated. By contrast, the universal set, is the set of all the candidates available vi the virtual table.
5	SelectionOperand	This is an abstract class/interface that must be implemented by any VT_Object that can act as the left hand side operand of the selection operator (the "." In "candidate.length"). This interface provides a means to get the result of applying the selection.

Table 6: Classes/Functions to make Implementation Easier and Efficient		
#	**Class**	**Description**
6	FilterOperand	This is an abstract class/interface that must be implemented by any VT_Object that can act as the left hand side of the selection operator, "[ ]". For example, given the expression X[Y], X must implement this interface. This interface provides a means to get the result of applying the filter to X.
	Function	**Description**
7	ApplySelection(X, Y)	A helper function used to simplify parsing of UAL. UAL expressions of the form X.Y are parsed to ApplySelection(X, Y). This function attempts to convert argument X to an instance of SelectionOperand and then invoke the method defined in SelectionOperand that yields the result applying selection Y on object X. Using the SelectionOperand interface in UAL, one can make selections like X.Y (i.e., select component Y) of X. In order for this to be meaningful, object X must implement the SelectionOperand interface.
7	ApplyFilter	This function is used for filter expressions of the formX[Y] and (all ~x in z where (y)) This is a helper function used to simplify parsing of UAL. UAL expressions of the form X[Y] are parsed to ApplyFilter(X,Y). This function attempts to convert argument X to an instance of FilterOperand and then invoke the method defined in FilterOperand that yields the result applying filter Y to object X.

Method of Operation

Referring now to Figure 1, there is shown a flow chart of a method for integrating heterogeneous information using a universal analysis language (UAL). The method begins by creating 102 a requirement or query. A requirement is an expression in UAL. The requirement may be created using an editor or other means. Thereafter, the method continues by parsing 104 the requirement by means of the parser.

After the requirement is parsed, the method continues by checking 106 or validating the results of the UAL parsing of the requirement. If the results are not correct, i.e. the requirement does not conform to the UAL grammar, the method is complete. Otherwise, the method continues by creating 108 an ordered sequence of evaluations, using the available evaluators, necessary to execute the requirement.

After the ordered sequence is created, the method continues by invoking 110 the evaluators to execute the evaluation sequence as determined in step 108. Thereafter, the results obtained of the evaluators are combined 112 using the appropriate UAL logic to produce the results of the requirement. The result is an ordering of decision candidates for the given requirement.

Operational Examples

Example Query

The following example illustrates how a sample query is executed. The query is represented in the system as a logical combination of criteria. The intent of the query is

for a user to find mutual funds based on their criteria. In this case the criteria (with their values) are:

1.	One Year Return,	*High*
2.	Risk Adjusted Return,	*Very High*
3.	Analysis,	*Long Term Capital Appreciation*
4.	Rating by people like me	*High*
5.	Similar To,	*KAUFX*

The first step is to determine an optimal evaluation sequence for these criteria based on the meta information available. The first criteria, for instance, uses the structured data access evaluator. The data relevant to One Year Return for each candidate is retrieved from the appropriate source and the evaluator compares this information to the definition of High. The structured data access evaluator accesses and evaluates structured database information. It takes a multivalued concept and returns the extent of the match between the concept and the actual information. "High" may be defined as "> 15%", in which case this comparison is trivial. Alternatively, it may be defined using truth functions such as those shown in Figure 2.

Alternatively, High may mean a better One Year Return than the average One Year Return of all the Funds known. Alternatively, it may mean better performance than, say, the S&P 500 index. The internal definition of the concept can be arbitrarily complex. This evaluator analyzes the extent to which the given, actual information matches the desired value.

For the criteria "long term capital appreciation", the specified evaluator may be a string similarity evaluator. This evaluator takes the argument string, and the source document for the candidate, which may be an analyst report, and returns an evaluation. This evaluation is converted into a truth value for this criterion, by looking up the rules for using this evaluator in the Universal Analysis Model. The criterion "Rating by people like me" employs a collaborative filter, such as Firefly or NetPerceptions' GroupLens. Similarly, depending on the execution sequence for the remaining criteria, each evaluation mechanism is invoked with the appropriate

parameters. Following this, the results from the evaluators are combined in accordance with their relationship as specified in the UAM, and the combined results, i.e. the ranked set of funds matching the query, are returned to the user. These results are documentable, so that a user is able to drill down into the rationale for the decision recommendation, all the way down to the actual information from the source. The sequence for executing a query is: (i) Determine an optimal execution sequence, (ii) Run the evaluators on the appropriate criteria, (iii) Use the various evaluators to convert the evaluators' results into the truth of the criteria, and (iv) Combine the results into a single value for each candidate, in this case mutual funds, given the entire query.

To keep the explanation simple, two assumptions are made in this query which the UAL does not require:

1. None of the criteria has any preference or importance, associated to it.
2. Each criteria is combined with an implicit AND connective.

Figure 3 illustrates a screen view of a query.

Criterion Definition Example 1

The example below defines a Criterion for One Year Return. The Criterion defines 4 values. All the values are evaluated with respect to a user's input. Thus a "good" one year return is between 0.75 to 1.25 times the input desired return with a drop-off of 10% of the desired value at the two extremes. Thus if the user input "10" as the input desired value, a good one year return is between 7.5% to 12.5%. Anything less than 6.5% is definitely not good and one year returns between 6.5% and 7.5% are good to some degree – "more good" as they approach 7.5%. In a similar manner, the definition includes other Criterion values such as "VeryGood", "Great", and "Extraordinary".

```
OneYearReturn (@desiredReturn : Float) : Fuzzy
{
    value Good ( ) { Pi(candidate.'One Year Return', @desiredReturn * 0.75,
                                                     @desiredReturn * 1.25,
                                                     @desiredReturn * 0.1,
                                                     @desiredReturn * 0.1)   }
```

```
value VeryGood ( ) {  Pi(candidate.'One Year Return', @desiredReturn * 1.35,
                                                       @desiredReturn * 1.75,
                                                       @desiredReturn * 0.1,
                                                       @desiredReturn * 0.1)  }

value Great ( ) {  Pi(candidate.'One Year Return', @desiredReturn * 1.85,
                                                   @desiredReturn * 2.5,
                                                   @desiredReturn * 0.1,
                                                   @desiredReturn * 0.1)  }

value Extraordinary ( ) { candidate.'One Year Return' > @desiredReturn * 2.5  }

}
```

Criterion Definition Example 2

In the following example, there is defined a Criterion that is selective in an inverse manner. It defines a Mediocre fund to be one whose "OneYearReturn" (the definition of Example 1 above) is "Good" or "Very Good" but not "Great" or "Extraordinary".

```
MediocreFund ( ) : Fuzzy
{
   (OneYearReturn for (Average(working set.'One Year Return')) is Good
    OR OneYearReturn for (Average(working set.'One Year Return')) is VeryGood)
   AND NOT (OneYearReturn for (Average(working set.'One Year Return')) is Great
            OR OneYearReturn for (Average(working set.'One Year Return')) is Extraordinary)
}
```

Criterion Definition Example 3

The following function uses text analysis to evaluate a Criterion. StringSimilarity returns a float number describes the degree to which the two input strings are similar. The result of StringSimilarity is fed to PI to return a fuzz result.

```
SeeksCapitalAppreciation ( ) : Fuzzy
{
  Pi(StringSim(candidate.Notes, "seeks capital appreciaiton"), 0.1, 1.0, 0.05, 0.0)
}
```

Universal Analysis Language (UAL) Grammar

The following is a BNF grammar for the Universal Analysis Language (UAL), according to one embodiment of the present invention.

```
//
// MACRO -- contains definitions of common types
// © 1998 by IBrain Software, Inc.
%macro

{squote}        "'";
{float}         '[0-9]+\.[0-9]+(e[\-+][0-9]+)?';
{truthValue}    '[10]\.[0-9]+[Tt]';
{integer}       '[0-9]+';
{simpleId}      '[A-Za-z_][A-Za-z_0-9]*';
{complexId}     '{squote}[A-Za-z_0-9][A-Za-z_0-9 ]*{squote}';

//
// EXPRESSION -- contains REGEXP definitions
//
%expression Main

'[ \t\n]+'                          %ignore;
'{simpleId}|{complexId}'            ID;
'\"[^\n\"]*\"'                      STRING_LITERAL;
'\('                                L_PAREN, '(';
'\)'                                R_PAREN, ')';
'\{'                                L_BRACE, '{';
'\}'                                R_BRACE, '}';
'\['                                L_BRACKET, '[';
'\]'                                R_BRACKET, ']';
':'                                 COLON, ':';
','                                 COMMA, ',';
'\|'                                VERT_LINE, '|';
'&'                                 AMPERSAND, '&';
'\*'                                STAR, '*';
'/'                                 SLASH, '/';
'\+'                                PLUS, '+';
'>'                                 GT, '>';
'<'                                 LT, '<';
'>='                                GE, '>=';
'<='                                LE, '<=';
'!='                                NE, '!=';
'='                                 EQUAL, '=';
```

```
'\-'                        MINUS, '-';
'%'                         PERCENT, '%';
'#'                         POUND, '#';
'!'                         EXCL, '!';
'\$'                        DOLLAR, '$';
'~'                         TILDA, '~';
'@'                         AT, '@';
'\.'                        PERIOD, '.';
{float}                     FLOAT_LITERAL;
{integer}                   INTEGER_LITERAL;
{truthValue}                TRUTH_LITERAL;
'[Aa][Nn][Dd]'              AND;
'[Oo][Rr]'                  OR;
'[Nn][Oo][Tt]'              NOT;
'[Ff][Oo][Rr]'              FOR;
'[Ii][Ss]'                  IS;
'[Aa][Ll][Ll]'              ALL;
'[Ii][Nn]'                  IN;
'[Ww][Hh][Ee][Rr][Ee]'      WHERE;
'value'                     VALUE;
'candidate'                 CANDIDATE;
'working set'               WORKING_SET;
'universal set'             UNIVERSAL_SET;
'row'                       ROW;
'with importance'           WITH_IMPORTANCE;

//////////////////////////////////////////////////////////////////
//
// PRECEDENCE
//
%prec

// Logical Ops (Lowest priority)
1, '&', %left;
1, '|', %left;


2, WITH_IMPORTANCE, %left;

// Logical NOT (higher than other logical ops)
3, NOT, %right;
3, '!', %right;

// Boolean comparison ops
```

```
4, '!=', %left;
4, '<', %left;
4, '>', %left;
4, '<=', %left;
4, '>=', %left;
4, '=', %left;

// Additive Ops
5, '+', %left;
5, '-', %left;

// Multiplicative Ops
6, '*', %left;
6, '/', %left;


////////////////////////////////////////////////////////////////////
//
// PRODUCTION -- grammar productions with start symbol = start
//
%production start

Start1              start -> criteria_definition;
Start2              start -> start criteria_definition;

//
// Criteria Definition
//
CritDef             criteria_definition -> identifier '(' typed_param_list ')' ':'
type_identifier criteria_body;

//
// Parameter Declaration List
//
TypedParamList      typed_param_list -> parameter_decl ',' typed_param_list;
TypedParamList1     typed_param_list -> parameter_decl;
TypedParamList0     typed_param_list -> ;

ParameterDec        parameter_decl -> '@' identifier ':' type_identifier;


//
// Criteria Body
//
CriteriaBody1       criteria_body -> '{' value_definition_list '}';
```

```
CriteriaBody2        criteria_body -> '{' expression '}';


//
// Value Definition List
//
ValueDefList value_definition_list -> value_def value_definition_list;
ValueDefList1        value_definition_list -> value_def;

ValueDef     value_def -> VALUE identifier '(' typed_param_list ')' '{' expression '}';


//
// Expression
//
Expression1          expression -> term;
ExpressionAnd        expression -> expression and term;
ExpressionOr         expression -> expression or term;

//
// Term
//
TermPlus             term -> term '+' term;
TermMinus            term -> term '-' term;
TermTimes            term -> term '*' term;
TermDivideBy         term -> term '/' term;
TermWIMP             term -> term WITH_IMPORTANCE term;
TermGT               term -> term '>' term;
TermLT               term -> term '<' term;
TermNE               term -> term '!=' term;
TermEQ               term -> term '=' term;
TermLE               term -> term '<=' term;
TermGE               term -> term '>=' term;
TermNot              term -> not term;
Term                 term -> literal;



//
// Literal
//
//LiteralNot         literal -> not literal;
LiteralFormula       literal -> formula;
LiteralExp           literal -> '(' expression ')';
```

```
//
// Formula
//
FormulaCrit        formula -> criteria_usage;
FormulaFunc        formula -> function_usage;
FormulaConst       formula -> constant_value;
FormulaSelect      formula -> field_expr;
FormulaParam       formula -> '@' identifier;

//
// Criteria Usage
//
CritUse1           criteria_usage -> identifier;
CritUse2           criteria_usage -> identifier FOR '(' argument_list ')';
CritUse3           criteria_usage -> identifier IS identifier;
CritUse4           criteria_usage -> identifier IS identifier '(' argument_list ')';
CritUse5           criteria_usage -> identifier FOR '(' argument_list ')' IS identifier;
CritUse6           criteria_usage -> identifier FOR '(' argument_list ')'
                                 IS identifier '(' argument_list ')';


//
// Field Expression -- an expression that selects/returns/operates on fields
//
FieldExprSel field_expr -> field_base field_selection;
FieldExprFunc      field_expr -> field_base field_func;
FieldExprIndex     field_expr -> field_base field_index;

FieldBaseCand      field_base -> CANDIDATE;
FieldBaseID  field_base -> '#' ID;
FieldBaseWS field_base -> WORKING_SET;
FieldBaseUS  field_base -> UNIVERSAL_SET;
FieldBaseFunc      field_base -> function_usage;
FieldBaseRow       field_base -> ROW;
FieldBaseParam     field_base -> '~' ID;
FieldBaseExpr      field_base -> field_expr;
FieldBaseFilter    field_base -> '(' ALL '~' ID IN WORKING_SET WHERE '('
expression ')' ')';

//
// Field Terms
//
FieldTermSel       field_selection -> '.' identifier;
FieldTermFunc1     field_func -> '(' argument_list ')';
```

```
FieldTermIndex1     field_index -> '[' expression ']';

//
// Function Usage
//
FuncUsage           function_usage -> identifier '(' argument_list ')';


//
// Argument List
//
ArgList             argument_list -> argument_list ',' expression;
ArgList1            argument_list -> expression;
ArgListEmpty        argument_list -> ;

//
// Binary Operators
//
//BOPTimes          binary_op -> '*';
//BOPDivide         binary_op -> '/';
//BOPPlus           binary_op -> '+';
//BOPMinus          binary_op -> '-';
//BOPWithImp        binary_op -> WITH_IMPORTANCE;
//BOPGT             binary_op -> '>';
//BOPLT             binary_op -> '<';
//BOPNE             binary_op -> '!=';

// More binary ops to be defined

//
// Type Identifiers
//
TypeIdentifier      type_identifier -> ID; // Need to add more flexible type ids


//
// Constant Value
//
ConstFloat          constant_value -> FLOAT_LITERAL;
ConstInt            constant_value -> INTEGER_LITERAL;
ConstString         constant_value -> STRING_LITERAL;
ConstTruth          constant_value -> TRUTH_LITERAL;


//
```

```
// Miscellaneous Productions
//
Identifier          identifier -> ID;
And1                and -> '&';
And2                and -> AND;
Or1                 or -> '|';
Or2                 or -> OR;
Not1                not -> '!';
Not2                not -> NOT;
```

The above description is included to illustrate the operation of the preferred embodiments and is not meant to limit the scope of the invention. The scope of the invention is to be limited only by the following claims. From the above discussion, many variations will be apparent to one skilled in the art that would yet be encompassed by the spirit and scope of the present invention.

What is claimed is:

Claims

1. A computer-implemented method for querying multiple different types of information, each type of information having a different evaluator, the method comprising:

receiving a query comprising an identification of at least two evaluators, at least one relationship between the evaluators, and a method of combining results from the evaluators;

parsing the query to create an evaluation sequence comprising an ordered sequence of invocations of the evaluators;

invoking the evaluators in the evaluation sequence; and

combining results from the evaluators according to the method of combining results from the evaluators specified in the query.

2. The method of claim 1, wherein the parsing step includes the substep of:

validating the query against a grammar;

wherein the invoking and combining steps are performed only if the query is successfully validated.

3. The method of claim 1, wherein the types of information are selected from the group consisting of structured information, unstructured information, qualitative information, and quantitative information.

4. The method of claim 1, wherein the evaluators are configured to perform a function from the group consisting of searching, prediction, collaborative filtering, and data mining.

5. A system for querying multiple different types of information, each type of information having a different evaluator, the system comprising:

a parser for receiving a query comprising an identification of at least two evaluators, at least one relationship between the evaluators, and a method of combining results from the evaluators;

means, coupled to the parser, for creating an evaluation sequence comprising an ordered sequence of invocations of the evaluators;

means, coupled to the creating means, for invoking the evaluators in the evaluation sequence; and

means, coupled to the invoking means, for combining results from the evaluators according to the method of combining results from the evaluators specified in the query.

6. The method of claim 5, the types of information are selected from the group consisting of structured information, unstructured information, qualitative information, and quantitative information.

7. The method of claim 5, wherein the evaluators are configured to perform a function from the group consisting of searching, prediction, collaborative filtering, and data mining.

8. A computer-readable medium having computer-readable program code modules embodied therein for querying multiple different types of information, each type of information having a different evaluator, the computer-readable medium comprising:

computer-readable program code modules configured to receive a query comprising an identification of at least two evaluators, at least one relationship between the evaluators, and a method of combining results from the evaluators;

computer-readable program code modules configured to parse the query to create an evaluation sequence comprising an ordered sequence of invocations of the evaluators;

computer-readable program code modules configured to invoke the evaluators in the evaluation sequence; and

computer-readable program code modules configured to combine results from the evaluators according to the method of combining results from the evaluators specified in the query.

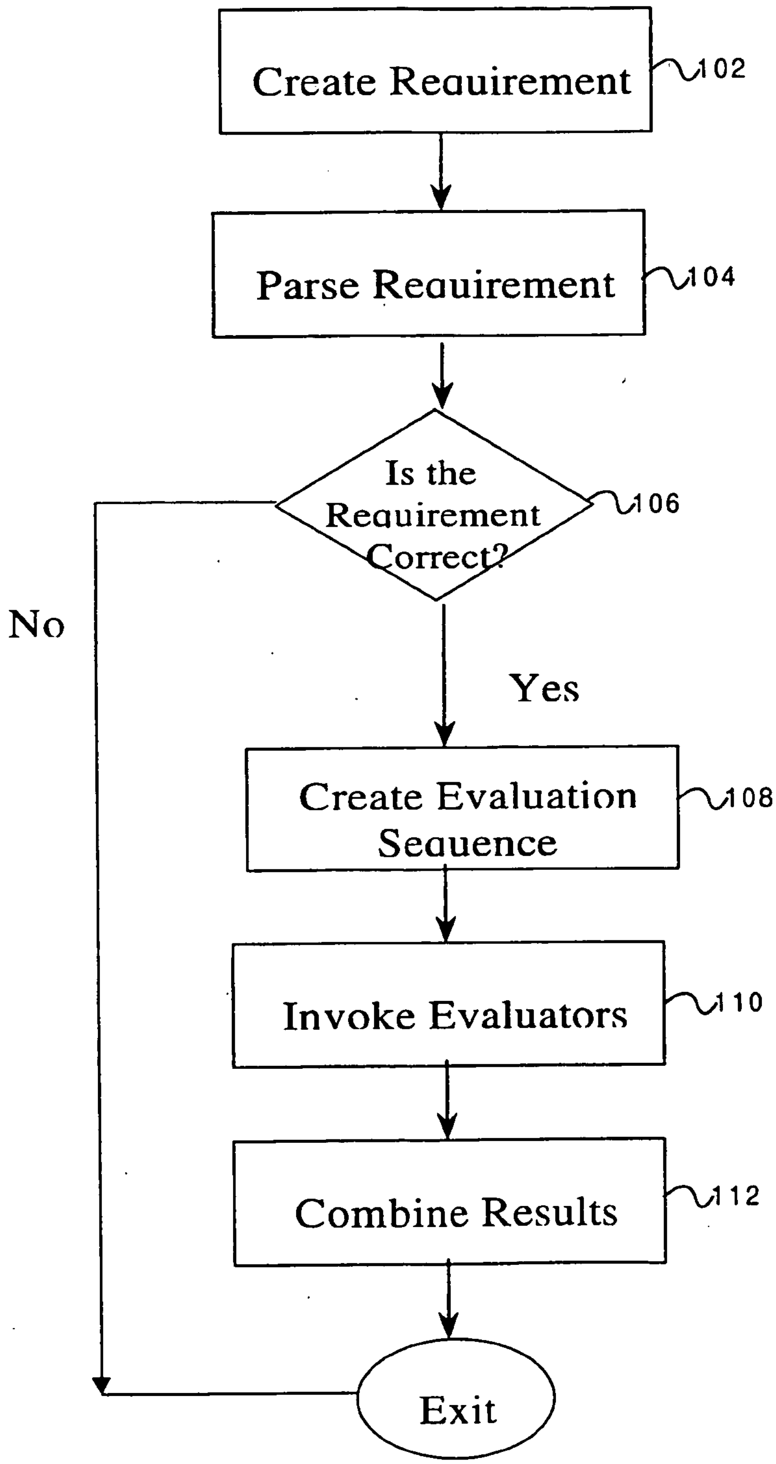
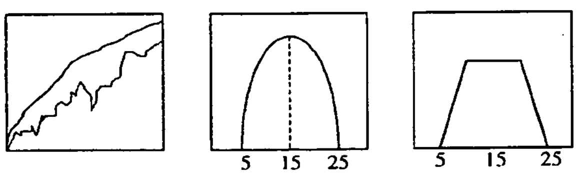

Create Requirement
102
Parse Requirement
104
Is the Requirement Correct?
106
No
Yes
Create Evaluation Sequence
108
Invoke Evaluators
110
Combine Results
112
Exit

Fig. 1




```
Candidate.1YrReturn
>
Average(Working Set.1YrReturn)
```

Fig. 2

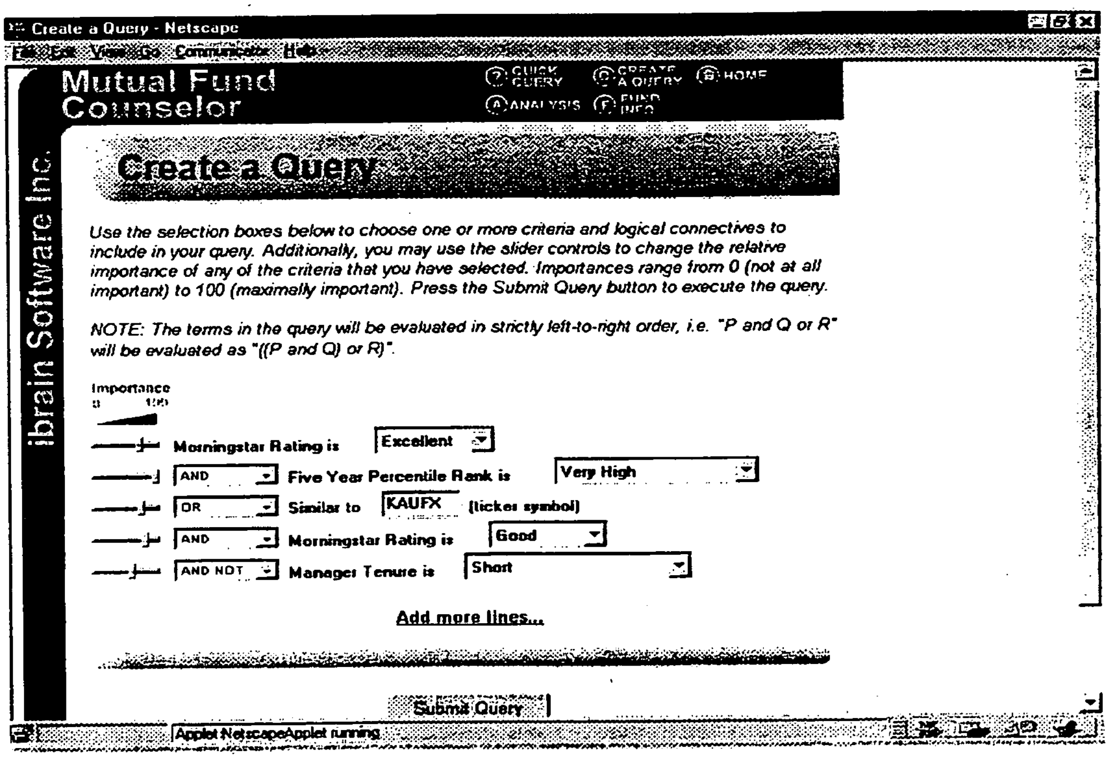

Create a Query - Netscape
Mutual Fund Counselor
ibrain Software Inc.
Create a Query
Use the selection boxes below to choose one or more criteria and logical connectives to include in your query. Additionally, you may use the slider controls to change the relative importance of any of the criteria that you have selected. Importances range from 0 (not at all important) to 100 (maximally important). Press the Submit Query button to execute the query.
NOTE: The terms in the query will be evaluated in strictly left-to-right order, i.e. "P and Q or R" will be evaluated as "((P and Q) or R)".
Importance
Morningstar Rating is Excellent
AND Five Year Percentile Rank is Very High
OR Similar to KAUFX (ticker symbol)
AND Morningstar Rating is Good
AND NOT Manager Tenure is Short
Add more lines...
Submit Query
Applet NetscapeApplet running

Fig. 3